National Défense
Defence nationale




# EVALUATION OF BORLAND TURBO VISION

by

David Lee and Andrew Mudry



## DEFENCE RESEARCH ESTABLISHMENT OTTAWA

TECHNICAL NOTE 93-37

Canada

December 1993
Ottawa

National Defence / Défense nationale

# EVALUATION OF BORLAND TURBO VISION

by

**David Lee and Andrew Mudry**
*Electronic Support Measures Section*
*Electronic Warfare Division*

**DEFENCE RESEARCH ESTABLISHMENT OTTAWA**
TECHNICAL NOTE 93-37

PCN
041LK

December 1993
Ottawa

## ABSTRACT

Turbo Vision™, a new application framework for PC DOS-based software development, is evaluated for its effectiveness as a base for computer applications. Turbo Vision provides an object-oriented, text-mode user interface and an event-driven program structure. An overview of the structure and software tools provided by Turbo Vision is presented. Various factors, such as consistency in architectural design, technical support, and ease of use, are considered.



## RÉSUME

Dans le présent rapport,Turbo Vision™, une nouvelle application destinée à l'élaboration de logiciels à base PC DOS, fait l'objet d'une évaluation visant à déterminer son efficacité en tant que base pour les applications informatiques. Turbo Vision fournit une interface centreé-objet ainsi qu'une structure de programme dirigée par les événements. Nous présentons une vue d'ensemble de la structure et des outils logiciels que fournit Turbo Vision. Divers facteurs, tels que la cohérence de la conception architecturale, les dispositifs techniques de soutien et la facilité d'utilisation sont également étudiés.

## EXECUTIVE SUMMARY

Turbo Vision™, a new application framework for PC DOS-based software development from Borland, is evaluated for its effectiveness as a base for computer applications. Turbo Vision provides an object-oriented, text-mode user interface and an event-driven program structure. Because of its tightly integrated, hierarchical architectural design and object-oriented implementation, Turbo Vision must be used in its entirety. It is not a disjoint set of tools. For the same reasons however, it is very flexible and easily extended with a minimal amount of code.

Turbo Vision provides a full suite of user interface features, such as windows, dialogue boxes, buttons, and keyboard and mouse support. Other important factors in evaluating Turbo Vision's usefulness include the learning curve, the level of technical support provided by Borland, the advantages and limitations resulting from using Turbo Vision's program structure, and the ease of conversion from the DOS-based, text mode environment to a Windows-based, graphics mode environment.

This document describes Turbo Vision, and evaluates it according to the above factors.

## TABLE OF CONTENTS

## LIST OF FIGURES

# 1.0 INTRODUCTION

Turbo Vision is an object-oriented, hierarchical framework for PC DOS based software applications. Turbo Vision provides a full suite of user interface features, ranging from simple buttons to a fully functional file selection module, that can be extended using the object oriented inheritance feature. Turbo Vision also provides an event driven program structure with keyboard and mouse support that can be likewise extended using object oriented techniques. Turbo Vision is available for use with Borland's C++ and Object Oriented Pascal compilers. Indeed, Borland uses Turbo Vision for its integrated development compiler environments. The C++ and Pascal versions are very similar in design, although only the C++ version 1.03 is discussed in this document.

In addition to the actual tools provided by a software package such as Turbo Vision, there are several other factors to consider. The primary consideration is the amount of time necessary to develop and to maintain an application. As an application's functionality is expanded following its initial development, the user interface and program structure must be easily extendable in order to handle any new modules or data flow paths. Since Turbo Vision imposes a program structure that is inextricably linked with its user interface features, this event-driven structure must also be evaluated. Other factors to consider are the level of technical support provided by Borland, the resources provided by third party sources, and the portability of applications developed for Turbo Vision. The observations and opinions presented in this document are based on personal experiences obtained during the development of a fax demodulator and viewer application in the Electronic Warfare Division at DREO. This example application is presented briefly, with focus on the influence of Turbo Vision on design decisions and program development.

# 2.0 AN OVERVIEW OF TURBO VISION

The elements of a Turbo Vision application can be separated into three groups: views, events, and mute objects. Views are rectangular program elements that access the screen display. Events are occurrences to which the application must respond, and finally, mute objects are any non-view objects.

## 2.1 Screen Display with Turbo Vision Views

A view is any element of the program that writes to the screen. Views include fields, scroll bars, labels, window borders, dialogue boxes, and even the application itself. They are derived from the Turbo Vision **TView** class. A view is responsible for a rectangular area of the screen and must manage the entire area. There are special collective views derived from the Turbo Vision **TGroup** class that own and manage arbitrary collections of views. Turbo Vision provides a skeleton application called **TApplication** which is a group view responsible for the entire screen and that owns, either directly or indirectly, every other view in the program. Since each view contains the information necessary to display itself, the program can redraw the entire screen simply by directing each view to draw itself in its own rectangular area. Thus, if the user

moves or resizes a window, the Turbo Vision framework automatically handles the redisplay of the screen. A **TGroup** based collective view also determines if any of its subviews are hidden by other views. It will only call the subviews which are not completely covered by other views.

The object oriented approach to this screen display framework considerably reduces the amount code that one must write to create new views. **TView** and **TGroup** member functions already provide all of the decision making code for scrolling, resizing, and moving windows, and for deciding when to redraw the screen and how to handle subviews. When one derives a new view from **TView** and possibly also from **TGroup**, this functionality is automatically inherited from Turbo Vision's base classes. All one really needs to supply is the member function for drawing the view. Turbo Vision handles the peripheral work of the user interface.

## 2.2 Turbo Vision's Event-Driven Framework

An event is any occurrence to which the application must respond. These include key presses, mouse clicks, commands from other parts of the program, or a character input from a serial port, Turbo Vision provides an event driven framework which provides centralized event gathering, event routing, and event queuing. One can concentrate on handling events instead of spending development time continually rewriting event gathering routines. As a user interface, Turbo Vision is responsible for obtaining keyboard and mouse inputs, as well as inputs from other sources that one defines. It is also responsible for sending these inputs, along with commands from other modules, to the target object or module.

Each event is stored in a Turbo Vision **TEvent** object that contains a header and data space for a message. There are four kinds of events: mouse events, keyboard events, nothing event, and message events. Mouse and keyboard events are generated by Turbo Vision from the corresponding user inputs. Nothing events are dead events that have already been completely handled. Message events, which are divided into commands, broadcasts, and user messages, are used for internal communication. Since there are a large number of predefined command and broadcast messages, as well as the possibility new user defined message events, Turbo Vision is flexible enough for large programs. There are predefined masks for quickly identifying the different kinds of events. Commands can also be individually disabled and enabled at any time. In general, the event format appears to be well-organized and flexible.

The Turbo Vision framework provides an event manager and a centralized mechanism for gathering these events from the event manager. One can override this event gatherer to search for events from other input sources, such as the serial port. Having a centralized event gatherer makes it easier to upgrade an application to work with a new input source.

Turbo Vision is responsible for routing the events to the proper module for processing. One can thus concentrate on handling the events without being concerned about getting the events or about from where the events came. The event driven framework is embedded within Turbo Vision's **TView** class, so it permeates the entire application framework. Class **TView** has the virtual member functions **getEvent()** and **handleEvent()** to get and handle events. One normally overrides **handleEvent()** to extend the capabilities of a new view. Due to the inheritance feature of object oriented programming, the derived view is capable of handling all of the events that **TView** handles, such as opening, closing, and resizing windows. One concentrates solely on the new functionality, allowing Turbo Vision's base classes to handle the basic functionality.

## 2.3 The Influence of Turbo Vision on Software Design

Turbo Vision offers some general software tools that are used internally and are also available to external applications. **TCollection** and various derivative classes provide basic mechanisms for handling collections of objects. The Turbo Vision stream manager provides an extension to C++ stream I/O. The Resource classes aid in storing and retrieving objects to and from streams.

The Collection classes provide dynamic data structures and member functions to handle objects such as strings, files, and resources. **TStringCollection** handles a sorted list of ASCII strings. **TResourceCollection** is derived from **TStringCollection**, and handles a collection of resources which are described in the following paragraphs. **TFileCollection** and **TDirCollection** are designed specifically to handle file names and DOS directory path names. **TFileList** and **TDirListBox** use these classes to display lists of files and directories through which the user can scroll and select individual entries.

A stream is an abstract data type representing a sequence of objects. The standard class **iostream** handles I/O operations in C++. In these operations, data which is stored in a file or a block of memory, sent to a monitor, read from the keyboard, or transmitted across a modem, is treated as a stream. Turbo Vision provides a stream manager which extends the capabilities of the C++ **iostream** class, allowing the handling of more complicated objects in Turbo Vision. The class **TStreamable** provides the read and write functionality that the stream manager requires. Since **TView** and **TCollection** are derived from **TStreamable** nearly all the classes in Turbo Vision, ranging from simple buttons and menus to extensive lists and even the entire application itself, are streamable. Each object inherits the streaming functionality from **TStreamable**, so it knows how to store its internal data on a stream and how to retrieve or initialize itself from a stream. Thus, the contents of one's entire application may be stored in a file with a single command that propagates through all the streamable objects in the application.

A Resource in Turbo Vision is an indexed random file. The Turbo Vision resource classes extend the file I/O capabilities of streams to provide more flexibility in accessing objects without knowing where the objects are located in a stream. Resource files can be used to customize an application without changing the code. For example displayed text, such as menu labels, on-line help, and dialog box information, can be stored in a file. To create a version in a different language, one only needs to replace the resource file instead of changing the original code and recompiling application. Configuration information, such as directory names, dialog box text, and colours, can be stored in a resource file for easy modification.

# 3.0 IMPORTANT CONSIDERATIONS IN SOFTWARE DESIGN

The main criteria in selecting an external software development package such as Turbo Vision is the cost associated with development and maintenance time. The greatest factor affecting this criteria is the architectural design of the software package. A well-designed, consistent architecture will lower the learning curve and reduce development and debugging time, while a poor disjointed design will force one to spend excessive amounts of time patching the software together. Various secondary factors such as the level of technical support provided and portability to other platforms must also be considered.

## 3.1 The Influence Of Turbo Vision On Software Design

A well-designed, consistent architecture adheres to the object-oriented programming methodology of structured, modular, reusable code that manages both the behavioural and informational complexity of an application. There should be a consistency among the modules and classes of an application in terms of naming conventions, and the partitioning of functionality among member functions.

The core of the application is that part of the application which does the actual work, such as signal processing or computations. The Turbo Vision architectural design separates the display (screen output) process and the input processes from the core of the application. The input processes may be external, such as keyboard and mouse inputs, or internal, such as commands and messages. This modularization provides for well-structured programs with clean internal interfaces, resulting in programs that are easier to debug and maintain. This enforced modularization has potential for positive long-term effects since it inhibits the degradation of an application's initial modularization which often occurs with successive maintenance upgrades.

In regards to development time, simple, straightforward user interfaces may be developed very quickly with minimal coding using Turbo Vision. One merely derives a new class based on the skeleton application **TApplication** and completes the application by adding member functions and by overriding **TApplication::handleEvent()** to control the member functions. To create a complete menu structure requires only one function call per menu entry, with all of these function calls linked together in a single call to the menu bar. Turbo Vision handles the formatting, hotkeys, pull down menus, and mouse support automatically. Indeed, one can construct straightforward applications simply by following the various example programs provided by Borland, without knowing much about the underlying structure.

Although Borland provides a large variety of example programs demonstrating the various features of Turbo Vision, one must eventually learn about Turbo Vision in order to use it effectively. Due to its tightly integrated structure, one must sometimes learn large portions of Turbo Vision in order to make extensions to the user interface. For example, suppose one wishes to modify the file lister **TFileList** to display information from each file, such as file size and date. One must examine the base classes **TSortedListBox**, **TListBox**, **TListViewer**, and **TView** to understand how **TFileList** works and then derive a new **TMyFileList** class that overrides key member functions of **TFileList**. Conversely, because Turbo Vision is object oriented, each derived class builds on its base classes, so learning about the base classes reduces the amount of learning in other derived classes. After learning about the base classes of **TFileList**, one also knows a great deal about the directory lister **TDirListBox**.

Ease and quality of application maintenance may be one of the most important features of Turbo Vision applications. Since Turbo Vision provides a consistent architecture, one can easily isolate the modules requiring modification when it is necessary to add or change the functionality of the application. For instance, in order for an object to handle new features, the object's **handleEvent()** member function must be altered to accept new commands or messages and to invoke the appropriate member functions implementing the new functionality. In order to display new information, one modifies the object's **draw()** member function. Moreover, since the Turbo Vision architecture is preserved throughout the application's lifetime, modifications maintain the separation of display and input processes from the core of each object and module. This inhibits the degradation of an application's initial modular structure through quick patches.

## 3.2 Technical Support and Other Resources

Another important factor to consider is the level of technical support provided for a commercial software package. As with the other tools provided with its language compilers, Borland is promoting and supporting Turbo Vision seriously. Borland provides a variety of technical notes on advanced topics, as well as technical support services. (See Appendix A.) They also provide the source code for Turbo Vision, an unusual and significant action. This is very valuable because, while the manual is well-written and provides a good introduction to Turbo Vision's main features, it is by no means exhaustive or sufficiently detailed for complicated extensions. Indeed, some of the low level system structure is only mentioned briefly. For example, to switch to graphics mode in order to display a bitmap image, one must suspend the Turbo Vision framework. Otherwise it would automatically switch the screen back to text mode at the first opportunity. Unfortunately, suspending the Turbo Vision framework also suspends the event manager, including keyboard and mouse support. One can implement keyboard and mouse support to handle this situation, but it is far easier to just re-enable and work directly with the event manager. The Turbo Vision manual discourages such practices, but it was found to reduce the amount of redundant code and development time during the programming of the application discussed in Section 4.0.

Third party resources for a software package are an important complement to the official technical support. Since Turbo Vision is quite new, very little external resources are currently available. Some shareware utilities for Turbo Vision, such as a dialogue box editor, have been developed. There is an unmoderated Internet news group established in Spring 1993 that has a small but consistent amount of traffic. Borland monitors the news group, although it does not provide technical support on the Internet. Freeware and shareware resources, which are listed in Appendix B, are available through two anonymous ftp sites. As of August 1993, no commercial utilities for Turbo Vision are known by the authors.

## 3.3 Portability To Other Functions

Another consideration in software design is that of portability. Turbo Vision is intended only for a PC MS-DOS system with text mode display. As is typical of all user interfaces, Turbo Vision uses low level software routines that are dependent on the operating system. Porting of a Turbo Vision application to another system would not be trivial. As an example of this, one could consider the porting of a Turbo Vision application from a PC DOS based system to a Microsoft Windows environment. ObjectWindows is a separate application framework created by Borland for MS-Windows systems. The authors have no personal experience in converting applications from Turbo Vision to ObjectWindows, but can offer a few simple observations. The Turbo Vision and ObjectWindows class hierarchies have a similar structure but significant differences are apparent. The **TView** class is essentially the root of all Turbo Vision classes. One usually overrides its **draw()** and **handleEvent()** member functions to provide the desired functionality and inherits the rest of the functionality unchanged. The **TWindowsObject** class is the corresponding base class for all the ObjectWindows classes, but it is completely different. **TWindowsObject** has different data members and member functions applicable to the Windows operating system. These fundamental differences propagate through the entire class hierarchy for both frameworks. These differences arise because, while Turbo Vision provides an event-driven, windowing framework, ObjectWindows provides an interface to the Windows

Application Programming Interface (API), which itself is a large user interface for Windows applications. It therefore is not straightforward to port an application from Turbo Vision to ObjectWindows.

## 4.0 AN EXAMPLE IMPLEMENTATION

Turbo Vision was used extensively in the development of the Defaxer application, a program for converting and viewing Group III fax transmissions. It serves as the C++ based user interface for the HIGHWIRE facsimile analysis system [2]. The application consists of four main modules: the user interface, the fax converter, the fax viewer, and the fax file manager. This section discusses the influence of Turbo Vision on the development of these modules.

### 4.1 The User Interface

The main menu in the Defaxer application appears as shown in Figure 1. The menu bar on the top line contains two menus, the File menu and the Configuration menu. The status line on the bottom shows quick keys for certain common commands. On the menu bar in the top right, the current time is displayed. The amount of available memory is displayed on the status line in the bottom right of the screen. Since all of these objects, as well as the background, are displayed on the screen, they are Turbo Vision views.

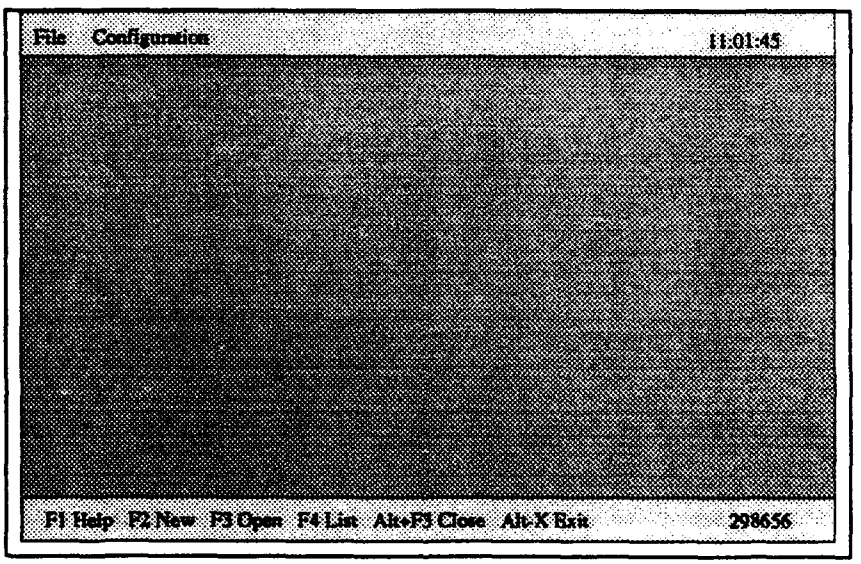


**Figure 1: Defaxer Application Main Menu**

Turbo Vision provides **TMenuBar**, **TSubMenu**, **TMenuItem**, and **TStatusLine** classes for the menu bar, submenus, menu items, and the status line. To create them, one simply initializes an object of the corresponding class with the appropriate information. Specifically, the following nested function call creates the Defaxer menu structure in its entirety:

```
return( new TMenuBar( r,
      *new TSubMenu( "~F~ile", kbAltF ) +
      *new TMenuItem( "~N~ew", cmNewFile, kbF2, hcNoContext, "F2" ) +
      *new TMenuItem( "~O~pen", cmFileOpen, kbF3, hcNoContext, "F3" ) +
      *new TMenuItem( "~L~ist", cmListFax, kbF4, hcNoContext, "F4" ) +
      newLine() +
      *new TMenuItem( "E~x~it", cmQuit, kbAltX, hcNoContext, "Alt-X" ) +
      *new TSubMenu( "~C~onfiguration", kbAltC ) +
      *new TMenuItem( "~F~ax directory", cmFaxDirCfg, kbNoKey ) +
      *new TMenuItem( "~V~ertical resolution", cmVertCfg, kbNoKey )
      ); .
```

Full mouse and keyboard input support is automatically provided by the Turbo Vision classes. The **TMenuBar** object handles the File and Configuration submenus. This means that the **TMenuBar** displays itself, places the submenu labels on the screen, and will open the appropriate submenu if the user clicks on its label with the mouse or presses the submenu hotkey. The hotkey is defined by the delimiting tildes (~). Similarly, the **TSubmenu** objects handle their menu items.

The status line shown in Figure 1 is created by the following nested function call:

```
return( statusLine = new TStatusLine( r,
      *new TStatusDef( 0, 0xFFFF ) +
      *new TStatusItem( "~F1~ Help", kbF1, cmHelp ) +
      *new TStatusItem( "~F2~ New", kbF2, cmNewFile ) +
      *new TStatusItem( "~F3~ Open", kbF3, cmFileOpen ) +
      *new TStatusItem( "~F4~ List", kbF4, cmListFax ) +
      *new TStatusItem( "~Alt-F3~ Close", kbAltF3, cmClose ) +
      *new TStatusItem( "~Alt-X~ Exit", kbAltX, cmQuit )
      );
```

In this case, the mouse and keyboard input support is provided by the Turbo Vision classes **TStatusLine**, **TStatusDef**, and **TStatusItem**. The **TStatusLine** displays itself, places the status item labels on the screen, and will invoke the appropriate item if the user clicks on its label. Each **TStatusItem** object will issue a message command if it is selected.

The time display at the top right of the screen, and the available memory indicator at the bottom right of the screen are objects derived from **TView**. They are non-active views that simply display their information whenever the Turbo Vision application is idle (i.e. not processing user inputs). They were developed quickly and easily from an example heap view program that Borland Technical Support provided (see Appendix B).

## 4.2 The Fax Converter

The fax converter converts fax image files from CCITT T.4 data format to black and white bitmap format [3]. It consists of an object of class **TConvertFax**, which accepts the input and output file names from the fax file manager. The **TConvertFax** object reads each fax image line from the input file, converts the input data into bitmap format, and stores the converted fax image line in the output file. The fax converter returns the number of fax image lines converted and the number of corrupted image lines. **TConvertFax** is not derived from any Turbo Vision class and operates independent of Turbo Vision. Thus Turbo Vision has no influence on this module.

The fax converter does however indirectly influence the display. During the conversion process, the user interface is inactive and the display is frozen. For a large input file corresponding to one or more highly detailed fax pages, the conversion process can take several seconds per page. Although not currently implemented, the fax converter can display the progress of the conversion and update it periodically. This can be accomplished easily by including the Turbo Vision TView class as one of the base classes of **TConvertFax**. **TConvertFax** will then have a **draw()** member function that displays the current number of lines converted and the current number of corrupted lines. During the conversion process, the fax converter will update the display each time it converts a certain number of lines.

## 4.3 The Fax Viewer

The fax viewer displays converted fax images on the screen. It can display the image at any integral scale factor. At higher resolutions it will allow the user to scroll through parts of the image. Turbo Vision provides full mouse and keyboard input.

The fax viewer consists of an object of class **TViewFax**, which accepts the input file name containing the fax image to be displayed. During initialization of the fax viewer, the screen is switched from text mode in which Turbo Vision operates, to graphics mode, which is necessary for displaying the bitmap image using Borland's graphics library. Because of a special feature of Turbo Vision, it was necessary to suspend the Turbo Vision display engine while operating the fax viewer.

Originally, **TViewFax** was derived from Turbo Vision's **TView**. Since the Turbo Vision display routine does not update any views completely covered by other views and since the **TViewFax** view covers the entire screen, none of the other views should be displayed if the Turbo Vision display engine is operating. Hence, the application should not return to text mode until the user exits the fax viewer. Unfortunately, Turbo Vision treats the status line differently, always updating the status line after every user input. Thus one cannot disable the display engine without suspending the entire Turbo Vision application engine.

Suspending the Turbo Vision application is not discussed in the User's Guide, although it is demonstrated in various example programs that Borland Technical Support provides (see Appendix B). Unfortunately, suspending Turbo Vision also eliminates the keyboard and mouse support. There are two options to deal with this problem: re-enable the keyboard and mouse support in Turbo Vision or develop keyboard and mouse support specifically for this module. In the Turbo Vision User's Guide, Borland discourages the former option and does not provide any information on how to accomplish it. However, the latter option contradicts the principle

of a consistent architectural design. With the latter option, the programmer must deal with two different lower level system calls during maintenance upgrades, an undesirable and costly complication. For this reason, it was decided that an attempt would be made to re-enable the keyboard and mouse support in the Defaxer application.

The re-enabling of Turbo Vision's event queue to provide the keyboard and mouse support, turned out not to be a complicated task. However, because Borland does not encourage this practice, learning how to accomplish it is a complicated task that involves investigating the source code provided by Borland. It should be noted that this code is sparsely documented and that it will not be serviced by Borland Technical Support except for bug fixes.

In summary, the viewing of a facsimile image in the Defaxer application is achieved by initiating a TViewFax object which switches the screen to graphics mode, suspends the Turbo Vision application engine to prevent it from returning to text mode, re-enables the event queue to provide mouse and keyboard support, and allows the fax viewer to operate. The fax viewer has a structure that parallels the Turbo Vision application, allowing it to handle events from the event queue in a manner consistent with the main application.

## 4.4 The Fax File Manager

The fax file manager constructs and maintains a list of previously stored data files. It uses certain Turbo Vision classes to display the list of fax files and to allow the user to select one of the images for display. The development of this module was greatly aided by the Turbo Vision classes for handling and displaying collections of files.

The list of fax files is stored in an object of class **TFaxCollection**, which is derived from **TCollection**. This list is displayed in a two column, scrollable box by an object of class **TFaxListBox**, which is derived from **TListBox**. **TFaxListBox** only needs to override **TListBox::getText()** member function to display specific information about each entry in the fax file list. Turbo Vision's **TListBox** handles all the other details of displaying the list, allowing the user to scroll through the list and to select a particular fax entry. The fax file manager dialogue box is illustrated in Figure 2.

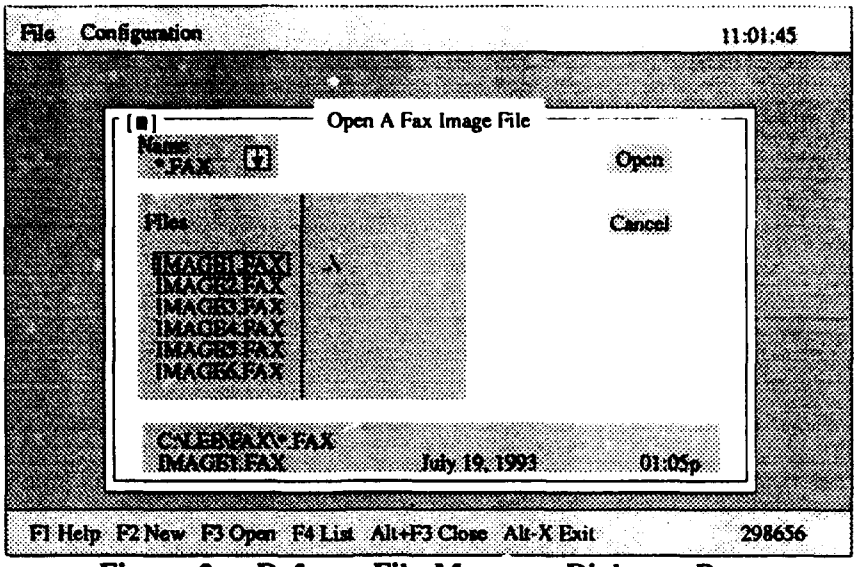


**Figure 2: Defaxer File Manager Dialogue Box**

## 5.0 CONCLUSIONS

The application framework Turbo Vision from Borland is an excellent software development framework for many PC DOS-based applications. It is appropriate for text mode applications written in C++ or Object Oriented Pascal. It is not appropriate for applications requiring extensive graphics displays. As with many user interfaces, it is designed for a single platform. It is therefore not easily ported to other platforms or operating systems. An example was considered of porting a Turbo Vision Application from a DOS environment to a MS-Windows environment. Technical support and supplementary resources for Turbo Vision are currently somewhat limited, but are still developing. Lack of detailed information on and support for the low level systems operation of Turbo Vision is the main drawback in developing applications that differ from the Turbo Vision application structure.

Turbo Vision was used extensively in the development of the Defaxer application. Basic user interface features such as menus, a file selector, dialog boxes, and a fax list box were developed quickly. The event driven structure of Turbo Vision isolated the user interface from the other modules and aided in the modularization of the application. Problems with operating in graphics mode to view fax images while using Turbo Vision's mouse and keyboard support were resolved with some effort.

## 6.0 REFERENCES

[1] Borland International, Inc. "Turbo Vision for C++ User's Guide", USA, 1992

[2] Mudry, A, Lee, D., Poulin, P. "The Feasibility Of Tactical VHF/UHF Facsimile ESM", DREO Report, To Be Published

[3] International Telegraph and Telephone Consultative Committee, CCITT Blue Book, Recommendation T.4

## APPENDIX A: TECHNICAL SUPPORT FOR TURBO VISION

Borland Technical Support provides technical support to registered Borland C++ users. Support for Turbo Vision is included since Turbo Vision is provided as a tool along with the compiler. Technical Support can be reached in a variety of ways, as listed in pp 5-7 of the Turbo Vision manual [1]. Borland provides technical information sheets on various topics as well as answering individual questions.

The following list is taken in part from the Borland Turbo Vision for C++ User's Guide, p. 6.

1. TechFax (800-822-4269 voice) is a 24-hour, automated service that sends technical information to one's fax machine free of charge. Use a touch-tone phone to receive up to three documents per call.

2. The Borland File Download Bulletin Board Service (408-439-9096 modem) has sample files, applications, and technical information for downloading via modem. No special setup is required.

3. CompuServe, GEnie, and BIX BBS's can receive technical support by modem.

4. Borland's mailing address is

   Borland International
   Technical Support Department - Turbo Vision
   1800 Green Hills Road
   P.O. Box 660001
   Scotts Valley, CA
   USA 95067-0001

5. Borland's phone number is 408-438-5300 (voice), available from 6:00 a.m. to 5:00 p.m. Pacific Standard Time.

6. Borland technical information is available on the anonymous ftp site *ftp.cica.indiana.edu* in the directory */pub/pc/borland*. Of special interest is a bug fix patch for version 1.03 available at this site in the file */pub/pc/borland/c/patch/bc31p1.zip*.

## APPENDIX B: THIRD PARTY RESOURCES FOR TURBO VISION

Since Turbo Vision is a recent product, third party resources are somewhat limited but still developing and expanding.

1. The Internet News group *comp.os.msdos.programmer.turbovision* is an unmoderated discussion group devoted to C++ and Turbo Pascal versions of Turbo Vision.

2. The anonymous ftp site *vtucs.cc.vt.edu* in the directory /turbo-vision maintains an archive of Turbo Vision messages, sample files, a bug list, a faq (frequently asked questions), freeware and shareware utilities, and peripheral information.

3. A shareware dialogue box editor is available at *vtucs.cc.vt.edu* in the file */turbo-vision/cpp/dlgdsc2.zip*. This editor allows one to place dialogue box objects in a dialogue box, to move them around, to adjust all the parameters of each object, and to generate the resulting code.

## DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Establishment sponsoring a contractor's report, or tasking agency, are entered in section 8.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
NATIONAL DEFENCE
SHIRLEY BAY, OTTAWA, ONTARIO K1A 0K2 CANADA

2. SECURITY CLASSIFICATION (overall security classification of the document. including special warning terms if applicable)

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C or U) in parentheses after the title.)

EVALUATION OF BORLAND TURBO VISION (U)

4. AUTHORS (Last name, first name, middle initial)

LEE, DAVID AND MUDRY, ANDREW H.

| 5. DATE OF PUBLICATION (month and year of publication of document) | 6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc.) | 6b. NO. OF REFS (total cited in document) |
|---|---|---|
| December 1993 | 13 | 3 |

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

DREO TECHNICAL NOTE 93-37

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include the address.)

DEFENCE RESEARCH ESTABLISHMENT OTTAWA
NATIONAL DEFENCE
SHIRLEY BAY, OTTAWA, ONTARIO K1A 0K2 CANADA

| 9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant) | 9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written) |
|---|---|
| 041LK | |

| 10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.) | 10b. OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor) |
|---|---|
| | |

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

- (XX) Unlimited distribution
- ( ) Distribution limited to defence departments and defence contractors; further distribution only as approved
- ( ) Distribution limited to defence departments and Canadian defence contractors; further distribution only as approved
- ( ) Distribution limited to government departments and agencies; further distribution only as approved
- ( ) Distribution limited to defence departments; further distribution only as approved
- ( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in 11) is possible, a wider announcement audience may be selected.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF FORM

13. ABSTRACT ( a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both offical languages unless the text is bilingual).

(U) Turbo Vision™, a new application framework for PC DOS-based software development, is evaluated for its effectiveness as a base for computer applications. Turbo Vision provides an object-oriented, text-mode user interface and an event-driven program structure. An overview of the structure and software tools provided by Turbo Vision is presented. Various factors, such as consistency in architectural design, technical support, and ease of use, are considered.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. *Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified.* If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

BORLAND C
OBJECT ORIENTED
MS DOS